**Exercise 2: E-commerce Platform Search Function**

**Understand Asymptotic Notation:**

**Explain Big O notation and how it helps in analyzing algorithms?**

Big O Notation is a way where the efficiency will be described in mathematical way.

Efficiency of an algorithm depends on time complexity and space complexity.

It helps in:

* Comparing the algorithms without running them.
* Acccording to contraints we can use the algorithm accordingly

**Types of data structures suitable for this problem**:

**Array**: It stores the collection of elements of homogeneous, in contiguous memery locations.

**HashMap**: It stores data as key pair values.

**TreeMap**: It stores keys in the sorted order.

**Trie**: fast prefix-based search

**Haep/PriorityQueue**: Search ranking or showing “Top N” product

**Implementation:**

**Main.java**

**package** com.EcommercePlatform;

//importing required modules for the program

**import** java.util.Arrays;

**import** java.util.Comparator;

**import** java.util.\*;

//Class is created with name Product

**class** Product {

**int** productId; // productId must be integer

String productName; //product name and category can be string

String category;

**public** Product(**int** id, String name, String cat) {

**this**.productId = id;

**this**.productName = name;

**this**.category = cat;

}

@Override

**public** String toString() {

**return** "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

//Two search functions were written for comparision

**class** SearchFunctions {

//Linear search function

**public** **static** Product linearSearch(List<Product> products, String name) {

**for** (Product product : products) {

**if** (product.productName.equalsIgnoreCase(name)) //the case must be ignored to avoid confusion

{

**return** product;

}

}

**return** **null**; //if the product not fount it will return null

}

//Binary search function

**public** **static** Product binarySearch(List<Product> products, String name) {

**int** l = 0, r = products.size() - 1; //l: left, r: right, m: mid, c: comparator

**while** (l <= r) {

**int** m = l + (r - l) / 2;

**int** c = products.get(m).productName.compareToIgnoreCase(name);

**if** (c == 0)

**return** products.get(m);

**else** **if** (c < 0)

l = m + 1;

**else**

r = m - 1;

}

**return** **null**;//if the product not fount it will return null

}

}

//Main class

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

List<Product> productsList = **new** ArrayList<>();

System.***out***.print("Enter no of entries of products: ");

**int** n = sc.nextInt();

sc.nextLine(); // consume newline

// Input details of products will be taken

**for** (**int** a = 0; a <= n-1; a++) {

System.***out***.println("\nEnter Product Details " + (a + 1));

System.***out***.print("Product ID: ");

**int** productId = sc.nextInt();

sc.nextLine(); // consume newline

System.***out***.print("Product Name: ");

String productName = sc.nextLine();

System.***out***.print("Category: ");

String productCategory = sc.nextLine();

productsList.add(**new** Product(productId, productName, productCategory));

}

System.***out***.print("\nEnter product name to search: ");

String searchName = sc.nextLine();

// linear search function calling

Product linearResult = SearchFunctions.*linearSearch*(productsList, searchName);

**if** (linearResult != **null**)

System.***out***.println("\n\*\*Linear Search\*\* Product Found:\n" + linearResult);

**else**

System.***out***.println("\n\*\*Linear Search\*\* Product not found.");

// binary search function calling

// Must sort before binary search

List<Product> sortedProducts = **new** ArrayList<>(productsList);

sortedProducts.sort(Comparator.*comparing*(p -> p.productName.toLowerCase()));

Product binaryResult = SearchFunctions.*binarySearch*(sortedProducts, searchName);

**if** (binaryResult != **null**)

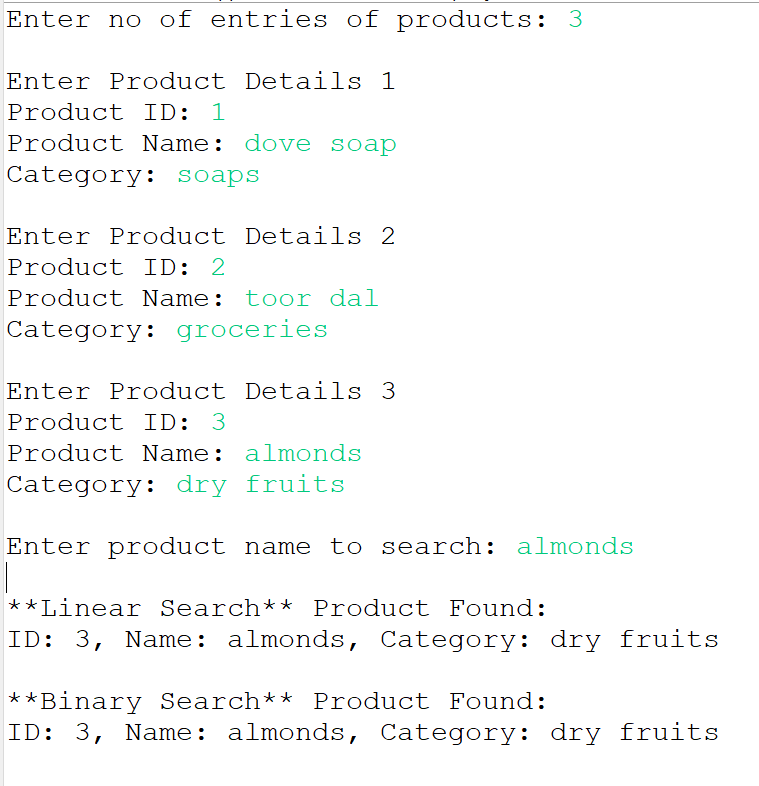
System.***out***.println("\n\*\*Binary Search\*\* Product Found:\n" + binaryResult);

**else**

System.***out***.println("\n\*\*Binary Search\*\* Product not found.");

}

**Output:**

}

**Analysis :**

**ArrayList :**

Add : O(1)

Update: O(n)

Delete: O(n)

Optimization :

* Use binary search on a sorted ArrayList to cut down search time to O(log n). However, insertion and deletion still take O(n) because of shifting.
* Use LinkedList if you often need to insert or delete, but you will lose index-based access.